# Content:

[Content: 1](#_Toc425377435)

[Exercise 1:Design a table (75') 2](#_Toc425377436)

[Exercise 2: Querying and Filtering Data (45') 5](#_Toc425377437)

[Exercise 3: Grouping and Summarizing Data (60') 13](#_Toc425377438)

# Exercise 1:Design a table (75')

**Q1**:Create the tables (with the most appropriate/economic field/column constraints & types).

|  |
| --- |
| CREATE TABLE TRAINEE(  TraineeID int IDENTITY(1,1) PRIMARY KEY,  Full\_Name nvarchar(50),  Birth\_Date date,  Gender bit,  ET\_IQ tinyint DEFAULT 0 CHECK (ET\_IQ >= 0 AND ET\_IQ <= 20),  ET\_GMath tinyint DEFAULT 0 CHECK (ET\_GMath >= 0 AND ET\_GMath <= 20),  ET\_English tinyint DEFAULT 0 CHECK (ET\_English >= 0 AND ET\_English <= 50),  Training\_Class varchar(30),  Evaluation\_Notes nvarchar(200)  ) |

**Q2**:Change the table TRAINEE to add one more field named Fsoft\_Account which is a not-null &unique field.

|  |
| --- |
| ALTER TABLE TRAINEE ADD Fsoft\_Account varchar(30) NOT NULL UNIQUE |

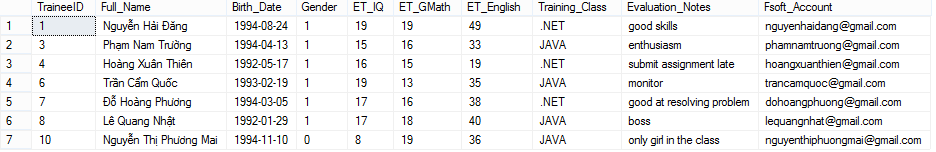
**Q3**: Add at least 10 records into created table.

|  |
| --- |
| SET DATEFORMAT dmy;  INSERT INTO TRAINEE(Full\_Name, Birth\_Date, Gender, ET\_IQ, ET\_GMath, ET\_English, Training\_Class, Evaluation\_Notes, Fsoft\_Account)  VALUES(N'Nguyễn Hải Đăng', '24/08/1994', 1, 19, 19, 49, '.NET', N'good skills', 'nguyenhaidang@gmail.com')  INSERT INTO TRAINEE(Full\_Name, Birth\_Date, Gender, ET\_IQ, ET\_GMath, ET\_English, Training\_Class, Evaluation\_Notes, Fsoft\_Account)  VALUES(N'Lê Quang Vinh', '26/08/1993', 1, 7, 6, 23, '.NET', N'take off', 'lequangvinh@gmail.com')  INSERT INTO TRAINEE(Full\_Name, Birth\_Date, Gender, ET\_IQ, ET\_GMath, ET\_English, Training\_Class, Evaluation\_Notes, Fsoft\_Account)  VALUES(N'Phạm Nam Trường', '13/04/1994', 1, 15, 16, 33, 'JAVA', N'enthusiasm', 'phamnamtruong@gmail.com')  INSERT INTO TRAINEE(Full\_Name, Birth\_Date, Gender, ET\_IQ, ET\_GMath, ET\_English, Training\_Class, Evaluation\_Notes, Fsoft\_Account)  VALUES(N'Hoàng Xuân Thiên', '17/05/1992', 1, 16, 15, 19, '.NET', N'submit assignment late', 'hoangxuanthien@gmail.com')  INSERT INTO TRAINEE(Full\_Name, Birth\_Date, Gender, ET\_IQ, ET\_GMath, ET\_English, Training\_Class, Evaluation\_Notes, Fsoft\_Account)  VALUES(N'Dư Phát Tài', '21/09/1994', 1, 15, 7, 27, 'JAVA', N'have good ideas', 'duphattai@gmail.com')  INSERT INTO TRAINEE(Full\_Name, Birth\_Date, Gender, ET\_IQ, ET\_GMath, ET\_English, Training\_Class, Evaluation\_Notes, Fsoft\_Account)  VALUES(N'Trần Cẩm Quốc', '19/02/1993', 1, 19, 13, 35, 'JAVA', N'monitor', 'trancamquoc@gmail.com')  INSERT INTO TRAINEE(Full\_Name, Birth\_Date, Gender, ET\_IQ, ET\_GMath, ET\_English, Training\_Class, Evaluation\_Notes, Fsoft\_Account)  VALUES(N'Đỗ Hoàng Phương', '05/03/1994', 1, 17, 16, 38, '.NET', N'good at resolving problem', 'dohoangphuong@gmail.com')  INSERT INTO TRAINEE(Full\_Name, Birth\_Date, Gender, ET\_IQ, ET\_GMath, ET\_English, Training\_Class, Evaluation\_Notes, Fsoft\_Account)  VALUES(N'Lê Quang Nhật', '29/01/1992', 1, 17, 18, 40, 'JAVA', N'boss', 'lequangnhat@gmail.com')  INSERT INTO TRAINEE(Full\_Name, Birth\_Date, Gender, ET\_IQ, ET\_GMath, ET\_English, Training\_Class, Evaluation\_Notes, Fsoft\_Account)  VALUES(N'Nguyễn Khoa Minh Nhân', '01/12/1993', 1, 6, 17, 17, '.NET', N'nothing', 'nguyenhkhoaminhnhan@gmail.com')  INSERT INTO TRAINEE(Full\_Name, Birth\_Date, Gender, ET\_IQ, ET\_GMath, ET\_English, Training\_Class, Evaluation\_Notes, Fsoft\_Account)  VALUES(N'Nguyễn Thị Phương Mai', '10/11/1994', 0, 8, 19, 36, 'JAVA', N'only girl in the class', 'nguyenthiphuongmai@gmail.com') |

**Q4**:Create a VIEW which includes all the ET-passed trainees. One trainee is considered as ET-passed when he/she has the entry test points satisfied below criteria:

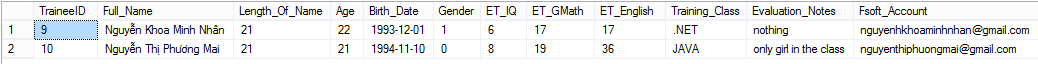
* ET\_IQ + ET\_Gmath>=20
* ET\_IQ>=8
* ET\_Gmath>=8
* ET\_English>=18

|  |
| --- |
| CREATE VIEW dbo.view\_PassEntryTest  AS  SELECT TraineeID,  Full\_Name,  Birth\_Date,  Gender,  ET\_IQ,  ET\_GMath,  ET\_English,  Training\_Class,  Evaluation\_Notes,  Fsoft\_Account  FROM TRAINEE  WHERE ET\_IQ + ET\_GMath >= 20  AND ET\_IQ >= 8  AND ET\_GMath >= 8  AND ET\_English >= 18 |



**Q5**:Query the trainee who has the longest name, showing his/her age along with his/her basic information (as defined in the table)

|  |
| --- |
| SELECT trainee1.TraineeID,  trainee1.Full\_Name,  LEN(trainee1.Full\_Name) AS Length\_Of\_Name,  DATEDIFF(year, trainee1.Birth\_Date, GETDATE()) AS Age,  trainee1.Birth\_Date,  trainee1.Gender,  trainee1.ET\_IQ, ET\_GMath,  trainee1.ET\_English,  trainee1.Training\_Class,  trainee1.Evaluation\_Notes,  trainee1.Fsoft\_Account  FROM TRAINEE trainee1  WHERE NOT EXISTS  (  SELECT 1  FROM TRAINEE trainee2  WHERE LEN(trainee2.Full\_Name) > LEN(trainee1.Full\_Name)  ) |



# Exercise 2: Querying and Filtering Data (45')

|  |  |
| --- | --- |
| **Query 1** | Write a query that retrieves the columns ProductID, Name, Color and ListPrice from the Production.Product table, with no filter. Your result set should look something like the following.  SELECT ProductID, Name, Color, ListPrice  FROM Production.Product  C:\Users\CHRIST\Desktop\Screenshots\Ex2-1.png |
|  |  |
| **Query 2** | Continue to work with the previous query and exclude those rows that are 0 for the column ListPrice. Your result set should look something like the following.  SELECT ProductID, Name, Color, ListPrice  FROM Production.Product  WHERE ListPrice <> 0 |
|  |  |
| **Query 3** | Use the same query, but this time you just want to see the rows that are NULL for the Color column. Your result set should look something like the following.  SELECT ProductID, Name, Color, ListPrice  FROM Production.Product  WHERE Color IS NULL |
|  |  |
| **Query 4** | Use the same query, but this time you just want to see the rows that *are not* NULL for the Color column. Your result set should look something like the following.  SELECT ProductID, Name, Color, ListPrice  FROM Production.Product  WHERE Color IS NOT NULL |
|  |  |
| **Query 5** | Now, combine two search arguments in the query you have been working with. You just want to see the rows that *are not* NULL for the column Color, and the column ListPricehas a value greater than zero. Your result set should look something like the following.  SELECT ProductID, Name, Color, ListPrice  FROM Production.Product  WHERE Color IS NOT NULL AND ListPrice > 0 |
|  |  |
| **Query 6** | Now we want a report that concatenates the columns Name and Color from the Production.Product table. Your result set should look something like the following. Make sure you exclude rows that are NULL for the column Color. Also notice the column name.  SELECT (Name + ': ' + Color) AS 'Name And Color'  FROM Production.Product  WHERE Color IS NOT NULL |
|  |  |
| **Query 7** | Customize the previous query so the answer looks like the following.  SELECT ('NAME: ' + Name + ' -- COLOR: ' + Color) AS 'Name And Color'  FROM Production.Product  WHERE Color IS NOT NULL |
|  |  |
| **Query 8** | Now we would like to see the columns ProductID and Name from the Production.Product table filtered by ProductID from 400 to 500. Write a query that makes your result set look something like the following. Try to make your WHERE clause as simple and readable as possible.  SELECT ProductID, Name  FROM Production.Product  WHERE ProductID >= 400 AND ProductID <= 500 |
|  |  |
| **Query 9** | We would like to see the columns ProductID, Name and color from the Production.Product table restricted to the colors black and blue. Write a query that makes your result set look something like the following. Try to make your WHERE clause as simple and readable as possible.  SELECT ProductID, Name, Color  FROM Production.Product  WHERE Color LIKE N'Black' OR Color LIKE N'Blue' |
|  |  |
| **Query 10** | **Wildcards**  This exercise and the next three following will make use of wildcards in Transact-SQL. To begin with, we would like a report on products that begins with the letter S.  Write a query that retrieves the columns Name and ListPrice from the Production.Product table. Your result set should look something like the following. Order the result set by the Name column.  SELECT Name, ListPrice  FROM Production.Product  WHERE Name LIKE 'S%'  ORDER BY Name |
|  |  |
| **Query 11** | Now we would like a report on products that begins with the letters S or A. Write a query that retrieves the columns Name and ListPrice from the Production.Product table. Your result set should look something like the following. Order the result set by the Name column.  SELECT Name, ListPrice  FROM Production.Product  WHERE Name LIKE 'S%' OR Name LIKE 'A%'  ORDER BY Name |
|  |  |
| **Query 12** | Adjust your query so you retrieve rows that have a Name that begins with the letters SPO, but is then *not* followed by the letter K. After this zero or more letters can exists. Order the result set by the *Name* column.  SELECT Name, ListPrice  FROM Production.Product  WHERE Name LIKE 'SPO[^K]%'  ORDER BY Name |
|  |  |
| **Query 13** | Write a query that retrieves *unique* colors from the table Production.Product. We do not want to see all the rows, just what colors that exist in the column Color. Your result set should look something like the following.  SELECT DISTINCT Color  FROM Production.Product |
|  |  |
| **Query 14** | Write a query that retrieves the unique combination of columns ProductSubcategoryID and Color from the Production.Product table. Format and sort so the result set accordingly to the following. We do not want any rows that are NULL.in any of the two columns in the result.  SELECT DISTINCT ProductSubcategoryID, Color  FROM Production.Product  WHERE ProductSubcategoryID IS NOT NULL AND Color IS NOT NULL  ORDER BY ProductSubcategoryID ASC, Color DESC |
| **Query 15** | Something is “wrong” with the WHERE clause in the following query.  We do not want any Red or Black products from any SubCategory than those with the value of 1 in column ProductSubCategoryID, unless they cost between 1000 and 2000.  SELECT ProductSubCategoryID  , LEFT([Name],35) AS [Name]  , Color, ListPrice  FROM Production.Product  WHERE Color IN ('Red','Black')  OR ListPrice BETWEEN 1000 AND 2000  AND ProductSubCategoryID = 1  ORDER BY ProductID  Write the query in the editor and execute it. Take a look at the result set and then adjust the query so it delivers the following result set.  SELECT ProductSubCategoryID  , LEFT([Name],35) AS [Name]  , Color, ListPrice  FROM Production.Product  WHERE Color IN ('Red','Black')  AND ProductSubCategoryID = 1  OR ListPrice BETWEEN 1000 AND 2000  ORDER BY ProductID | |
|  |  |
| **Exercise 16** | Use the Production.Product table to return product name, color and list price for each product. For the color column, where there is NULL, replace it with the string *Unknown*.  SELECT Name, ISNULL(Color, 'Unknown') AS Color, ListPrice  FROM Production.Product |

# Exercise 3: Grouping and Summarizing Data (60')

This exercise performs on AdventureWorks2008 database that included in the same folder with the assignment

|  |  |
| --- | --- |
| **Query 1** | How many products can you find in the Production.Product table? Your result set should look like the following.  SELECT COUNT(\*)  FROM Production.Product |
|  |  |
| **Query 2** | Write a query that retrieves the number of products in the Production.Product table that are included in a subcategory. The rows that have NULL in column ProductSubcategoryID are considered to not be a part of any subcategory.  SELECT COUNT(ProductSubcategoryID) AS HasSubCategoryID  FROM Production.Product |
|  |  |
| **Query 3** | How many Products reside in each SubCategory?  The answer to this is retrievable if you write a query that use the COUNT aggregate function combined with a GROUP BY clause.  The column ProductSubcategoryID is a candidate for building groups of rows when querying the Production.Product table. Your result set should look something like the result below.  SELECT ProductSubcategoryID, COUNT(Name) AS CountedProducts  FROM Production.Product  GROUP BY ProductSubcategoryID |
|  |  |
| **Query 4** | Try to write two different queries to find out how many products that do not have a product subcategory. One query without the WHERE clause and one query using a WHERE clause. The rows that have NULL in column ProductSubcategoryID are considered to not be a part of any subcategory.  SELECT COUNT(\*) - COUNT(ProductSubcategoryID)  FROM Production.Product    SELECT COUNT(\*) AS NoSubCat  FROM Production.Product  WHERE ProductSubcategoryID IS NULL |
|  |  |
| **Query 5** | A report is needed, the summary of products in stock. Write a query against another table this time, the Production.ProductInventory table.  SELECT ProductID, SUM(Quantity) AS TheSum  FROM Production.ProductInventory  GROUP BY ProductID |
| **Query 6** | Continue to write on the query in previous exercise. Add a WHERE clause that extracts the rows that have the column LocationID set to 40 and limit the result to include just summarized quantities less then 100.  SELECT ProductID, SUM(Quantity) AS TheSum  FROM Production.ProductInventory  WHERE LocationID = 40  GROUP BY ProductID  HAVING SUM(Quantity) < 100 |
|  |  |
| **Query 7** | In this query we also want to see what shelf the product is to be delivered from. Add code to the previous query.  SELECT Shelf, ProductID, SUM(Quantity) AS TheSum  FROM Production.ProductInventory  WHERE LocationID = 40  GROUP BY ProductID, Shelf  HAVING SUM(Quantity) < 100 |
|  |  |
| **Query 8** | We would like to see the average quantity for products where column LocationID has the value of 10. The table Production.ProductInventory has the answer.  SELECT AVG(Quantity) AS TheAvg  FROM Production.ProductInventory  Where LocationID = 10 |
|  |  |
| **Query 9** | To continue to write on the previous query, we would like to see the result by shelf excluding rows that has the value of N/A in the column Shelf. We also want to see a total average based on shelf only and also for all products (“grand total”).  SELECT ProductID, Shelf, AVG(Quantity) AS TheAvg  FROM Production.ProductInventory  WHERE LocationID = 10 AND Shelf <> 'N/A'  GROUP BY ROLLUP (Shelf, ProductID)  ORDER BY Shelf |
|  |  |
| **Query 10** | We want to know number of members (rows) and average list price in the Production.Product table. This should be grouped independently over the Color and the Class column. We are not interested in any rows where Color nor Class are null (WHERE Class IS NOT NULL AND Color IS NOT NULL).  Hint: think GROUPING SETS.  SELECT Color ,Class ,COUNT(\*) AS TheCount ,AVG(ListPrice) AS AvgPrice  FROM Production.Product  WHERE Class IS NOT NULL AND Color IS NOT NULL  GROUP BY GROUPING SETS (Color, Class) |
|  |  |
| **Query 11** | We now want to examine the function GROUPING. The following query generates the result below the query itself. Take a look and complete the query so it results to the second result set.  SELECT ProductSubcategoryID  , COUNT(Name) as Counted  FROM Production.Product  GROUP BY ROLLUP (ProductSubcategoryID)  Below you find the second result set, continue to write a complete query so it gives the following answer. The added column is for clarity regarding NULL.  SELECT ProductSubcategoryID, COUNT(Name) as Counted, GROUPING(ProductSubcategoryID) AS IsGrandTotal  FROM Production.Product  GROUP BY ROLLUP (ProductSubcategoryID)  ORDER BY ProductSubcategoryID |